FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Programação Visual para a definição de um estúdio televisivo na cloud baseado na tecnologia IP

António Presa

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Luís Teixeira

Proponente: Pedro Ferreira

18 de Julho de 2017

Programação Visual para a definição de um estúdio televisivo na cloud baseado na tecnologia IP

António Presa

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Maria Teresa Andrade
Arguente: Christophe Soares
Vogal: Luís Filipe Teixeira

---

18 de Julho de 2017

Resumo

Atualmente temos assistido a uma grande migração de uma vasta variedade de serviços televisivos para tecnologias IP e *Cloud*. Esta migração surgiu da necessidade de uma otimização destes serviços.

Otimização a nível de escalabilidade, a nível de fiabilidade, a nível monetário, a nível de conveniência. No entanto, é imperativo manter ou até melhorar a qualidade e fiabilidade destes serviços.

Nesta dissertação, o serviço em questão, é um estúdio televisivo. Um estúdio televisivo é algo extremamente complexo que envolve ter controlo sobre uma variedade de câmaras, microfones, iluminações, monitores, auscultadores. Associar tecnologias IP a todos estes periféricos significa que cada um destes vai ter um IP associado e que vão ter que comunicar entre si.

O foco desta dissertação é, então, o desenvolvimento de uma interface que permita a um utilizador, intuitiva, simples e rapidamente desenvolver o seu estúdio televisivo. Isto implica desenvolver uma página Web, onde o utilizador poderá, intuitivamente, criar e associar módulos de forma a criar o seu estúdio televisivo personalizado. Estes módulos representam *switches* e periféricos que também terão parâmetros importantes associados, como por exemplo, operador, largura de banda necessária, posição no espaço, para onde devem estar orientados (no caso das câmaras, microfones e iluminação), etc. Devido a limites de processamento dos servidores, limites de largura de banda, limites de RAM, e outros limites físicos, será necessário impor limites à criação do utilizador de modo a que não haja falhas ou quebras.

Para efeitos de teste e validação, foram realizados de testes A/B.

Concluindo, esta dissertação baseia-se no estudo e desenvolvimento de uma interface que permita a um utilizador, intuitivamente, criar e monitorizar os *workflows*, desde os periféricos até ao sinal de output final, um estúdio televisivo personalizado aos seus interesses de forma fácil e rápida.

Abstract

Nowadays, we have been witnessing a vast migration of a wide variety of television and broadcasting services to IP and Cloud technologies. This migration arose from the need for optimization and update of these services.

This optimization is referring to scalability, monetary savings and convenience. However, it is imperative to maintain or even improve the quality and reliability of these services.

In this dissertation, the service in question is a television studio. A television studio is something extremely complex that involves having control over a variety of cameras, microphones, lightings, monitors, headphones. Associating IP technologies to all these peripherals means that each of these will have an associated IP and they will have to communicate with a central server.

The focus of this dissertation is the development of an interface that allows a user, with or without any experience in informatics, to develop his own television studio. This consists of creating a web service, where the user can intuitively create and associate modules to create their own personalized TV studio. These modules represent switches and peripherals that will also have important associated parameters, such as required bandwidth, position in space, where they should be oriented to (in case of cameras, microphones and lighting). Due to server's processing limits, bandwidth limits, RAM limits, and other physical limits, it will be necessary to impose limits on a user creation so that there are no failures, freezes or breaks.

For testing and validation purposes, A/B testing was done.

In conclusion, this dissertation consists on the study and development of an interface that allows a user to intuitively create and control a television studio customized to their interests in an easy and fast way.

Agradecimentos

Queria, aqui, agradecer ao meu orientador, professor Luís Teixeira que esteve sempre disponível quando necessário e pelo apoio durante toda a dissertação. Também queria agradecer ao José Matias e Victor Fernandes, que, como *FrontEnd Developers* deram-me a conhecer novas tecnologias, e opinaram sobre o projeto, dando-me novas ideias tanto de funcionalidades como melhorias de *user experience*. Um obrigado também ao Pedro Santos e Pedro Ferreira por me ensinarem como um estúdio televisivo funciona. Por fim, um obrigado à MOG Technologies, onde esta dissertação foi desenvolvida, e a todos os que nela trabalham pelo ambiente e espírito de equipa.

António Presa

"Design can be art. Design can be asthetics. Design is so simple, that's why it is so complicated'

Paul Rand

Conteúdo

Lista de Figuras

Abreviaturas e Símbolos

IP	Protocolo de Internet
mDNS	multicast Domain Name System
RAM	Memória de Acesso Aleatório
SaaS	Service as a Service
SDI	Serial Digital Interface
UI	User Interface
UX	User Experience
Web	World Wide Web
ZB	ZettaByte
DOM	Document Object Model
MVP	Model-View-Presenter
MVC	Model-View-Controller
MVVM	Model–View–Viewmodel
JSX	JavaScript XML

Capítulo 1

Introdução

Cada vez mais verifica-se uma migração de serviços outrora físicos para software, o chamado *Software as a Service* (*SaaS*). Estes serviços estão a ser, cada vez mais, hospedados na *Cloud*. Isto causa um aumento de tráfego que se prevê que por 2020 terá multiplicado por 3.7 (de 3.9 ZB por ano em 2015 para 14.1 ZB em 2020) [Net16]. Por *Cloud* entende-se o processo de partilha de recursos de uma rede de computadores e servidores para cada indivíduo obter melhor performance. Hospedar serviços na *Cloud* significa que estes podem ser utilizados remotamente através da Internet, sendo que, por exemplo, um utilizador deixa de estar limitado pelas especificações e requisitos do seu computador para ter acesso a estes serviços. [PS09] Uma maneira de utilizar estes serviços é através de uma interface Web, no entanto, estas interfaces podem facilmente tornar-se muito complexas devido à possível vasta informação que um serviço requer de um utilizador. Logo, a dificuldade de implementar uma interface passa pela análise de uma abstração em que o utilizador não se aperceba da complexidade do serviço. [OMO11]

Em pouco tempo, as empresas de *broadcast* aperceberam-se das imensas vantagens de trazer estas tecnologias para o mundo da televisão, logo começaram a estudar possíveis alternativas de implementar estas tecnologias nos seus estúdios. No entanto, isto traz algumas dificuldades a serem exploradas posteriormente neste documento.

Contudo, este documento irá focar-se mais no estudo e criação de uma interface que permita a estas empresas criarem e gerirem os seus estúdios baseados nestas tecnologias.

1.1 Contexto/Enquadramento

Esta dissertação encontra-se inserida num projeto, em ambiente empresarial, que tem como objetivo a implementação de mecanismos que possibilitem a criação de um estúdio televisivo na *Cloud* para eventos em direto. De forma a explicar de forma prática, utilizando um exemplo real de um concerto, em que existem diversos periféricos (câmaras de vídeo), cada uma com um IP único, que transmitem diretamente para um estúdio através de uma rede ou da Internet, que está

localizado remotamente, onde existe *software* capaz de recolher toda esta informação, editá-la e fazer *broadcast* para os consumidores finais.

Neste projeto existe uma outra tese, desenvolvida por Vasco Gonçalves e com o nome "Orquestração e aprovisionamento de um estúdio televisivo baseado na tecnologia IP na Cloud"[Gon17], que interpreta o JSON providenciado por esta interface e, lança (virtualiza) e gere os componentes do estúdio televisivo na *network* de forma autónoma.

A empresa em que esta dissertação está inserida é a MOG Technologies. A MOG está inserida no mercado televisivo e desenvolve soluções que permitem a interoperabilidade entre equipamentos utilizados por estúdios televisivos.

1.2 Motivação e Objetivos

Com esta evolução surgem novas possibilidades e, neste caso mais especificamente, as evoluções de redes IP (tornaram-se mais rápidas e estáveis) e tecnologia *Cloud* (virtualização de componentes outrora em *hardware*) tornou teoricamente possível a implementação de um estúdio televisivo a funcionar apenas com base nestas tecnologias que trazem imensas vantagens (a serem discutidas no próximo capítulo). Algumas empresas da indústria, como por exemplo a BBC, têm vindo a dar passos neste sentido e algumas até estão próximas de soluções finais, no entanto, estas ainda contêm falhas. Esta dissertação tem como objetivo o estudo e o desenvolvimento de uma interface que possibilite ao utilizador, profissional de um estúdio, criar o seu próprio estúdio televisivo de um modo simples, eficaz e intuitivo.

1.3 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais 4 capítulos. No capítulo 2, é descrito o estado da arte e são apresentados trabalhos relacionados. No capítulo 3, é apresentada uma solução para o problema assim como tecnologias a serem utilizadas. No capítulo 4, é apresentada a solução desenvolvida assim como um exemplo de um caso de utilização. No capítulo 5 é apresentado o trabalho futuro, juntamente com um plano de trabalho.

Capítulo 2

Revisão Bibliográfica

2.1 Introdução

Neste capítulo vamos explorar o estado de arte de estúdios de televisão, como estes funcionam e transmitem informação. Será abordado como aplicar tecnologias IP a estes estúdios e suas vantagens e desvantagens. O desenvolvimento de interfaces Web e como tornar estas intuitivas e apelativas para o utilizador também será discutido. Estes tópicos serão importantes para o desenvolvimento da solução proposta, a abordar posteriormente.

2.2 Um estúdio televisivo

Atualmente, cerca de 25 horas por semana são despendidas, por indivíduo, a ver televisão [RUS15], o que torna a televisão uma parte significativa no quotidiano das pessoas. Normalmente, o maior número de espetadores concentra-se entre as 20 e as 22 horas, seguindo em segundo lugar o intervalo das 11 às 16 horas. Isto implica que nestes intervalos é imperativo que as estações de televisão não sofram falhas, já que quando estas ocorrem, os espetadores, inconscientemente, são levados a trocar de canal. Esta responsabilidade de não falhar cai, maioritariamente, nos estúdios televisivos.

Um estúdio televisivo é uma instalação onde se recebe, edita e transmite sinais de televisão, respetivamente. Primeiro recebem-se vários sinais de áudio e vídeo (normalmente apresentados em monitores para os técnicos e representantes estarem a par da emissão) diferentes, que por sua vez estão ligados a mesas de misturas (analisar figura 2.1).

Figura 2.1: Exemplo de uma mesa de mistura juntamente com ecrãs de monitorização

A ideia básica de uma mesa de mistura é permitir a um profissional responsável escolher por que canal de vídeo ou áudio optar que, por sua vez, transmite um sinal para os espetadores. Contudo, permite muito mais que isto. Permite ao profissional editar os sinais originários para criar uma melhor experiência ao espetador. Por exemplo, é possível criar ciclos, em que se está sempre a repetir o mesmo sinal (útil quando ocorre uma falha no sinal principal); criar transições entre dois sinais com os mais variados efeitos; criar e editar *overlays*, isto é, inserir imagens ou vídeos por cima do sinal principal, é comum utilizar logos da empresa e informação (ver figura 2.2). Estas mesas permitem que as edições sejam todas realizadas em tempo real e com o mínimo possível de latência, o que é útil para eventos em direto.

Figura 2.2: Exemplo de *overlays*

Na figura anterior os *overlays* estão representados no canto superior esquerdo e na parte inferior.

Desta mesa, obtemos como resultado, um sinal de saída. Este sinal é emitido então para o espetador. É possível transmitir este sinal de três formas diferentes: *broadcast*, satélite e cabo. Na primeira, o sinal é transmitido por ondas rádio. No caso de satélite, o sinal é emitido para um satélite, que por sua vez emite de volta para a Terra, onde os espetadores têm parabólicas para captar o sinal. A outra alternativa para transmitir o sinal é através de cabo, em que existe um cabo (normalmente de cobre ou fibra ótica) que liga a estação televisiva a todas as casas dos espetadores. [Tec16]

2.2.1 Transmissão gravada

Nesta forma de transmissão, o sinal, ou sinais, é proveniente de um dispositivo de armazenamento onde já existem os ficheiros de vídeo e áudio guardados. Um profissional do estúdio até pode fazer as edições que deseja aos ficheiros e criar um ficheiro final, que pode ser gravado e colocado num horário e transmitido automaticamente sem ter que o profissional esteja presente.

A principal vantagem deste tipo de transmissão é que existe uma menor pressão sob os profissionais do estúdio. Também é útil para a entidade televisiva reduzir custos, já que pode ser criado um calendário automático de programas a serem reproduzidos, normalmente em horários em que o número de espetadores seja mínimo, e minimizar o número de profissionais no estúdio.

2.2.2 Transmissão ao vivo

Este é o tipo de transmissão onde ocorrem mais falhas. Aqui, tanto os sinais de vídeo como de áudio são captados, em tempo real, por uma ou mais câmaras e microfones, e transmitidos para o estúdio.

Atualmente existem duas alternativas para transmitir estes sinais: diretamente por cabos *SDI*, ou satélite, dependendo da distância do evento ao estúdio. Caso o evento esteja a ser filmado no próprio estúdio, utilizam-se cabos SDI ligados diretamente das câmaras à mesa de mistura (ver figura 2.3).

Figura 2.3: Exemplo de um cabo *SDI*

Caso o evento seja num lugar remoto utilizam-se carrinhas-estúdio (ver figura 2.4 como intermediários entre as câmaras e o estúdio. Estas carrinhas são um estúdio móvel, sendo que é nestas onde o sinal é editado localmente [BBD+04]. As câmaras estão, na mesma, ligadas por cabos SDI, no entanto, neste caso estão ligadas à carrinha e não ao estúdio, que por sua vez transmite o sinal via satélite para o estúdio e este apenas tem que o transmitir para os espetadores.

Figura 2.4: Exemplo de uma carrinha-estúdio

2.2.3 Transmissão ao vivo sobre IP e *Cloud*

Uma alternativa recente à transmissão em direto em locais remotos é utilizando tecnologias IP e *Cloud*. Associar estas tecnologias a um estúdio televisivo é uma tarefa árdua, no entanto extremamente vantajosa. [BRWT13]

Por um lado, a *Cloud* permite-nos virtualizar componentes e ligações. Isto é, componentes que existam atualmente num estúdio televisivo, como por exemplo, um *video switcher*, que é o que nos

permite escolher de entre os sinais de vídeo de entrada, um para a saída. Virtualizar componentes significa que estas podem encontrar-se implementadas remotamente tanto por *software* como por *hardware*.

Para isto ser possível é necessário a existência de uma rede na qual todos os periféricos (câmaras, microfones e as componentes abordadas no parágrafo anterior) estejam conectados. Isto implica que têm que ser compatíveis com IP. Cada dispositivo tem um id único que permite a comunicação entre cada periférico e o equipamento escolhido. Também é preciso um software capaz de gerir o estúdio. Parte deste software será discutido posteriormente, na solução proposta.

Nisto existem algumas dificuldades e algumas restrições da tecnologia em si, como por exemplo: como TCP é assíncrono [Jor05], é impossível prever exatamente quando uma informação específica vai chegar e o *delay* que esta traz consigo; também é preciso ter em atenção a largura de banda da rede.

Também temos que especificar alguns fatores que são impossíveis de eliminar. Primeiro, temos que assumir que, por mínima que seja, poderá existir sempre perda de informação [Jor05]. Temos ainda que assumir que existirá sempre algum atraso na transferência de informação, pelo que, se é impossível ser eliminado, a segunda melhor opção é fazer com que esse atraso seja quantificável.

Portanto, uma possível opção para corrigir o atraso da informação imprevisível é criar um *buffer*, isto é, um recipiente que permite ir guardando informação à medida que for transferida. Após um atraso que definimos começamos a ler a informação do *buffer*. Assim conseguimos corrigir variações de velocidade na transferência de informação. Também temos que ter excecional atenção à largura de banda da rede, já que as câmaras ao transmitirem informação estão a ocupar parte da rede, chamada largura de banda, e esta rede tem uma capacidade máxima que não pode ser ultrapassada. Todos estes aspetos são obrigatórios estar presentes no software que controla o estúdio para que este funcione corretamente.

Dito isto, as maiores vantagens de utilização destas tecnologias são que a indústria televisiva consegue diminuir consideravelmente os custos e aumentar a flexibilidade na criação de estúdios remotos [dCP16]. É possível diminuir o custo na medida em que as Carrinhas-estúdio tornam-se obsoletas (cada uma destas pode custar vários milhões de euros [Jac14]) porque, por um lado, as componentes que estão atualmente presentes nestas são passíveis a serem virtualizadas e, por outro, como as câmaras podem transferir diretamente, ou através de uma *gateway* para estas componentes virtualizadas, estas carrinhas não são necessárias para transmitir o sinal das câmaras por satélite para o estúdio. Também é possível aumentar a flexibilidade na criação de um estúdio visto que deixam de existir limitações a nível de hardware uma vez que estas componentes transitam para *software*. Para além do que não temos limitações do número de sinais de entrada, pelo que apenas é necessário adicionar uma nova câmara (por exemplo) à rede para começar a transmitir.

Um bom exemplo destas vantagens é o caso dos jogos olímpicos do Brasil no ano de 2016. Neste evento foram utilizadas 52 carrinhas-estúdio [Ser16], o que seria apenas necessário uma com um servidor para onde as câmaras podessem comunicar.

2.3 Interfaces de utilizador

Atualmente o uso da Internet é cada vez mais comum, o que torna o estudo das interfaces de utilizador cada vez mais importantes. Uma *UI* é o meio de comunicação entre um computador e um utilizador. É através destas que um utilizador comunica um pedido (*input*) e recebe o resultado (*output*). Existem várias opções de comunicação entre o utilizador e o computador, por exemplo através do toque, audição, visualização, fala, escrita. [Gal07]

Existem os mais variados tipo de interfaces, no entanto, nesta secção apenas serão abordadas interfaces de aplicações Web. É entendido por aplicação Web, uma aplicação onde todas ou algumas partes sejam descarregadas da Web cada vez que é executada e apresentada ao utilizador por recurso a um *browser*. Estas têm dois lados: o lado do cliente (executado num *browser*) e o lado do servidor (num lugar remoto), e predominantemente existem dois tipos de aplicações Web: aplicações apenas executadas no lado do servidor onde o lado do utilizador serve apenas para *inputs* e apresentação de resultados; e aplicações executadas tanto no lado do cliente como no do servidor ou apenas no utilizador. Estas últimas estão limitadas em termos de processamento, pelo computador do utilizador , que varia de computador para computador.

No desenvolvimento de interfaces existem vários aspetos a ter em conta. Por exemplo, um investigador tentou otimizar uma interface melhorando a legibilidade e clareza, tornando-a menos sobrecarregada de informação. Após testes de utilizadores, o investigador chegou à conclusão que houve um aumento de 20% de produtividade, os utilizadores demoraram 25% menos tempo para completar uma tarefa e que os utilizadores cometeram 25% menos erros. [Gal07]. O que nos leva à conclusão que menos é mais. Alguns destes aspetos serão discutidos na subsecção abaixo.

2.3.1 Usabilidade vs Experiência do utilizador (*UX)*

Embora relacionados, usabilidade e *UX* têm diferentes significados. Apesar de ambos serem importantes para o desenvolvimento de uma interface, enquanto usabilidade está relacionada com a habilidade em realizar uma tarefa fácil e intuitivamente, experiência do utilizador está relacionada com o sentimento que provem em realizar a mesma tarefa.

Em termos práticos, pode dizer-se que existe um alto nível de usabilidade numa interface quando: um utilizador requer pouco esforço mental para realizar uma tarefa, existe uma menor frequência de erros em *inputs* do utilizador e quando esta requer pouco tempo a ser percebida/entendida pelo utilizador.

Por outro lado, *UX* é uma categoria mais qualitativa do que a usabilidade, uma vez que é baseada em sentimentos do utilizador e intuição [Gar10], o que a torna mais árdua tanto para desenvolver como para avaliar.

Ambas são um pouco subjetivas, no entanto muitos *delevopers* focam-se apenas na usabilidade, assumindo que é mais importante que experiência de utilizador, e negligenciam esta última [Gar10], o que diminui a qualidade geral da interface uma vez que as duas estão sempre conectadas.

2.3.2 Testes e validação

É importante testar se o produto não tem erros e também se é válido, o que não é a mesma coisa. Ser válido implica que obedeça aos requisitos, objetivos definidos para esta anteriormente.

Verificar se uma interface contem erros é uma tarefa difícil e exaustiva [ADJ+11], no entanto, temos os nossos objetivos do que testar previamente definidos. Para diminuir, até certo nível, a dificuldade em testar uma interface podemos recorrer a *frameworks* automáticas, como por exemplo a explicada em [ADJ+11].

Testar se uma interface tem erros é um processo praticamente direto, apesar de difícil, já que sabemos pelo que procurar, como vimos, no entanto, testar a validade de uma interface é uma tarefa mais complexa, visto que uma possível parte dos requisitos passam por fatores subjetivos.

2.3.2.1 Testes A/B

Uma possível opção de testar a validade de uma interface é através de testes A/B. Estes testes consistem no desenvolvimento de duas interfaces muito parecidas, apenas com diferenças subtis, como por exemplo na figura 2.5. Num grupo controlo de utilizadores, a maioria vai, subconscientemente, optar por uma das interfaces, mesmo que não consigam identificar diferenças entre as duas (explicado na subsecção Usabilidade vs Experiência do utilizador (*UX*)), o que torna uma das interfaces mais apelativa do que a outra para a maioria dos utilizadores. Logo, é por esta que devemos optar [Opt17].

Figura 2.5: Exemplo de um teste A/B onde a variação A obteve mais sucesso que a variação B. [VWO17]

Na imagem acima, taxa de conversão significa a percentagem dos utilizadores que realizaram a ação pretendida exposta na variação, por exemplo carregar num *link*.

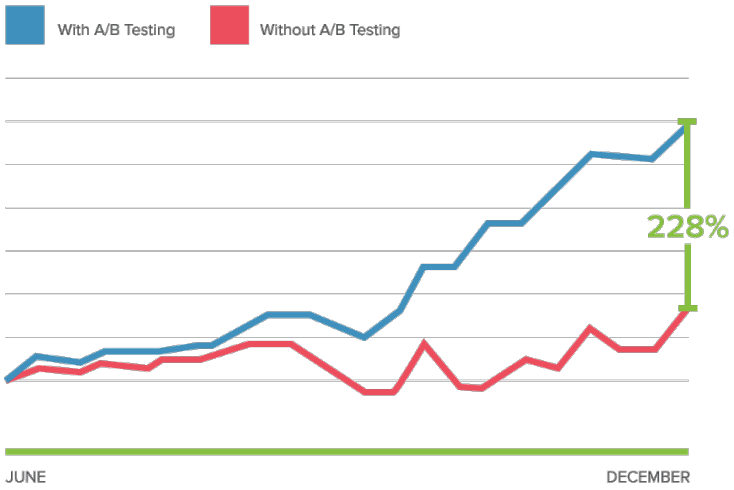

Figura 2.6: Vantagem da utilização de testes A/B. [Opt17]

A imagem acima refere-se à variação da taxa de conversão dos utilizadores que visitam um *website*. A linha a azul corresponde à interface otimizada através destes testes, enquanto a vermelha corresponde à mesma interface que não foi otimizada.

2.4 Tecnologias Web

Existem inúmeras tecnologias para desenvolver aplicações web, vamos analisar algumas nas subsecções abaixo. Todas elas com um propósito diferente e desenhadas com um objetivo especifico em mente. Umas criadas para proporcionarem rápidas velocidades de renderização, outras com facilidade e velocidade de implementação em vista. Isto leva a que um *developer* tenha que fazer um estudo cuidado sobre quais os objetivos que tem para a aplicação ou ferramenta Web e tenha que chegar a uma conclusão sobre a que tecnologias recorrer. Para tal, nos subcapítulos a seguir serão explicadas algumas destas tecnologias.

2.4.1 *Frameworks* vs *Libraries*

Quase todos os meses são criadas novas *frameworks* e *libraries* [Bub16] de Javascript.

A definição de uma *framework* é algo um pouco subjetivo, pelo que varia de *framework* para *framework*, no entanto, a maioria segue um padrão. Sendo este uma implementação de MV* (*Model-View-**), como, por exemplo, MVC, MVP, entre outros.

Models, ou modelos, em Javascript correspondem aos dados da aplicação, dados estes que tanto podem ser pedidos ao lado do servidor guardados numa base de dados como podem estar do lado do cliente (fotos, utilizadores, valores).

Views, ou vistas, podem ser tão simples como um conjunto de *templates*, como podem conter toda a lógica, roteamento e a ligação dos dados dependendo da *framework*. Em ambos os casos, o papel destes é manter o estado do *Document Object Model* (DOM).

O asterisco, corresponde à parte flexível da definição de uma *framework*. Este pode ser uma camada que comunica com as *views* e os *models* mantendo-os atualizados e sincronizados tanto por modificações nas *views* como nos *models*. Também poderá ser nesta camada onde é gerido o roteamento da aplicação.

Por outro lado, uma *library*, ou biblioteca, é apenas uma coleção de classes. Bibliotecas têm como vantagem primordial a reutilização de código.

O seguinte diagrama representa a relação entre *frameworks* e bibliotecas.

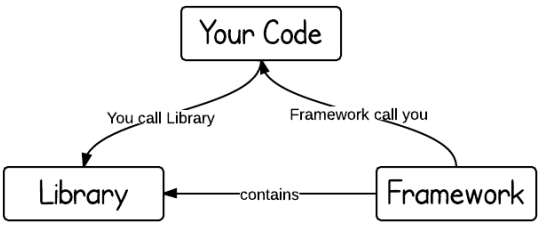

Figura 2.7: Relação entre *libraries* e *frameworks* [lib11]

Enquanto que uma *framework* chama o código desenvolvido quando necessário, uma biblioteca é chamada pelo código - o chamado *Inversion of Control*, ou inversão de controlo.

2.4.2 React vs Angular2 vs Backbone

Neste subcapítulo serão abordadas vantagens, desvantagens e o objetivo principal de cada uma destas *frameworks*/*libraries*.

2.4.2.1 React

Foi inicialmente desenvolvido pelo Facebook em 2013 e tem como objetivo principal ser implementada em grandes aplicações *web* onde o conjunto de dados esteja constantemente a mudar. Ao contrário das outras 2 tecnologias, esta é uma biblioteca e não uma *framework*.

Um dos pontos positivos principais desta biblioteca é a existência de um DOM virtual. Isto implica que quando existe uma alteração é criado um DOM virtual que é comparado com o DOM real (o que o utilizador está a ver) e caso existam alterações entre os dois, então partes do DOM real são substituídas pelas partes necessárias do DOM virtual. Nisto resulta um aumento de velocidade e eficiência de atualização. Também, em termos de velocidade de renderização, React permite que esta seja realizada do lado do servidor, (uma outra vantagem de renderização no lado do servidor é que permite o salto para *mobile development* muito facilmente). Um outro ponto positivo é a existência da sua própria extensão: JSX. Esta tem como objetivo tornar o processo de *coding* mais intuitivo. Isto devido a que tem uma sintaxe muito semelhante à do HTML, difere apenas, por

exemplo, nas palavras reservadas, como *class* é uma palavra reservada e uma *tag* HTML pode conter classes, então criaram o *className* que se usa nas *tags* HTML em vez de *class*.

Um dos objetivos principais desta biblioteca é tornar o desenvolvimento de interfaces gráficas orientado a componentes, tornando assim muito fácil a reutilização de código. Os componentes de React têm a particularidade de terem um *scope* isolado, desta forma facilmente é possível reutilizar de componentes criados e disponibilizados pela comunidade de React.

2.4.2.2 Angular2

Angular foi inicialmente criado em 2011 pela Google. Depois, em 2016 foi lançada uma segunda versão. Nesta nova versão, os *developers* da Google vieram ao encontro do React e trouxeram os níveis de velocidade de renderização para os níveis próximos do React (o que não acontecia com a primeira versão) através dos *Watchers* que também adicionaram capacidade de renderização do lado do servidor. No entanto Angular2 usa typescript ao contrário de React que usa JavaScript.

Como Angular2 é uma framework torna a sua implementação e configuração mais fácil e rápida porque, como já tem uma arquitetura pré-definida, não tem o *developer* que perder tempo a estudar uma arquitetura para a sua aplicação.

No entanto, Angular2 tem uma API enorme, o que também a torna de difícil aprendizagem.

2.4.2.3 Backbone

Backbone foi apresentado em 2010. É a *framework* mais leve e mais pequena aqui apresentada, o que a torna mais fácil de aprender. É do tipo MVC e é mais usada para manipulação de dados e *requests* tanto entre os controladores e as *views* como entre os serviços e o servidor. No entanto, Backbone tem algumas desvantagens que o tornaram menos apelativo em relação à concorrência e, por isso, entrou um pouco em desuso. Algumas desvantagens do Backbone são, por exemplo, como esta *framework* é muito leve, tem poucas funcionalidades nativas, o que requer, por parte do *developer*, escrita de mais código.

2.4.3 Redux

O objetivo do Redux é simplificar as comunicações entre componentes diferentes. Para isto, em vez de componentes comunicarem entre si diretamente, é criada uma *Store* onde é guardado todo o estado do sistema.

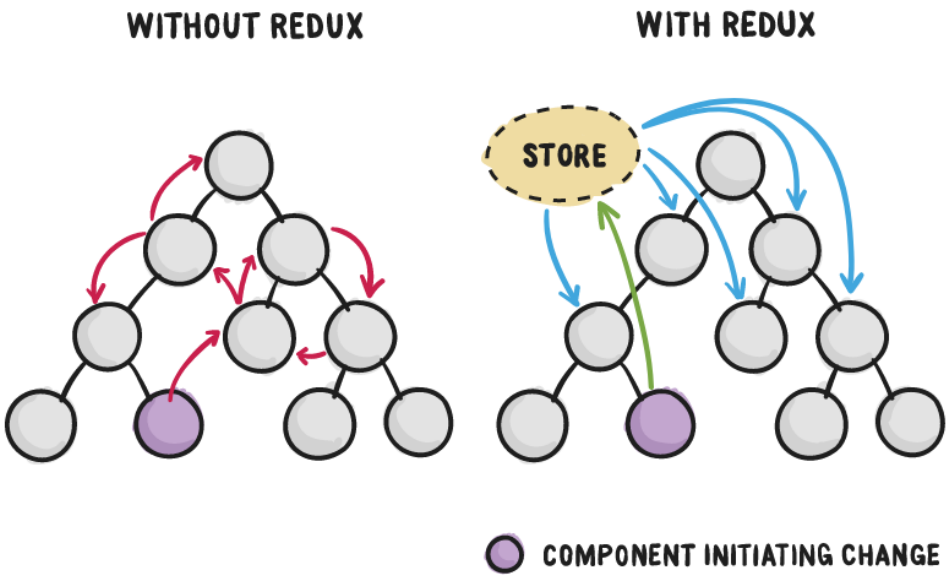

Figura 2.8: Exemplo da estrutura de funcionamento do Redux [css17]

Uma componente faz um *dispatch* para a *store* para atualizar esta com um novo estado e as componentes que tenham alguma dependência com a componente iniciante fazem *subscribe* à *store* para se manterem atualizadas. A utilização de Redux torna o fluxo de dados mais legível e simples.

2.4.4 CSS3 vs SCSS vs SASS

SASS, ou *Syntactically Awesome StyleSheets*, é uma extensão de CSS que adiciona funcionalidade e elegância a este. A diferença entre SASS e SCSS (*Sassy CSS*) é meramente em termos de sintaxe. SCSS é uma verão mais recente de SASS.

Algumas das funcionalidades que esta nova linguagem introduz ao CSS são:

1. Existência de variáveis, o que torna o código mais legível e fácil de editar (ver figura 2.9)

2. É compatível com todas as versões de CSS

3. É mais estável e elegante aquando a criação de CSS, o que torna a sua produção mais rápida (por exemplo, ver figura 2.12

4. Possibilita a existência de *nested classes*, ou classes encapsuladas (ver figura 2.10)

```scss
// define variables
$red: #f00;
$wk: -webkit-;

// use variables
.link:hover {
    color: $red;
    #{$wk}transition: all 1s ease;
}
```

```css
.link:hover {
    color: #f00;
    -webkit-transition: all 1s ease;
}
```

Figura 2.9: Exemplo do recurso a variáveis. Do lado esquerdo temos o SCSS, enquanto do lado direito temos o CSS [fut14]

```scss
ul {
    padding: 0;
    li {
        list-style: none;
        a {
            color: blue;
            &:hover {
                color: lime;
            }
        }
    }
}
```

```css
ul {
   padding: 0;
}

    ul li {
        list-style: none;
    }

        ul li a {
            color: blue;
        }

            ul li a:hover {
                color: lime;
            }
```

Figura 2.10: Exemplo de *nesting*. Do lado esquerdo temos SCSS, enquanto do direito CSS [fut14]

```scss
%box {
    margin: 2px;
    border: 1px solid black;
}

.greenBox {
    @extend %box;
    border-color: lime;
}

.greenBoxBlueBackground {
    @extend .greenBox;
    background-color: blue;
}
```

```css
.greenBox, .greenBoxBlueBackground {
    margin: 2px;
    border: 1px solid black;
}

.greenBox, .greenBoxBlueBackground {
    border-color: lime;
}

.greenBoxBlueBackground {
    background-color: blue;
}
```

Figura 2.11: Exemplo de heranças de classes. Do lado esquerdo verificamos SCSS, enquanto do direito verificamos CSS [fut14]

No entanto é necessário compilar SASS e SCSS para CSS. Neste caso será compilado através do Webpack (será abordado posteriormente).

2.4.5 ES6 vs JS

JS, ou JavaScript foi inicialmente desenvolvido em 1995 por Brendon Eich. Numa fase inicial era chamado de Mocha. Depois foi oficialmente renomeado de LiveScript, que por sua vez, quando a empresa onde Eich trabalhava (Netscape) e a Sun uniram-se, o nome foi mais uma vez alterado para JavaScript.

Posteriormente, a Netscape decidiu entregar o JavaScript à Ecma International para definir um *standard* e guiar o caminho do JavaScript. Daí surgiu o nome EcmaScript. As versões de EcmaScript 1 à 3 foram lançadas entre 1997 e 1999 com as funções básicas de uma linguagem, expressões regulares e *error handling*. Depois foi iniciado o desenvolvimento do EcmaScript 4, no entanto o comité TC39 (*Ecma Technical Committee 39*), onde estava Eich, decidiu abandoná-lo devido a divergências no conjunto de *features* a serem desenvolvidas e acabaram por criar um EcmaScript 3.1.

Mais tarde, em 2009, surge o EcmaScript 5 que mudou o modo como vemos o JavaScript, com o *strict mode*, suporte de JSON, *getters* e *setters*, e novos métodos para arrays como por exemplo o *forEach* e *filter*.

Em 2015 surge finalmente o EcmaScript 2015, ou ES6. Esta versão traz com ela inúmeras melhorias, como por exemplo: *arrow function*, *typed arrays*, *promises*, *Block scoping variable and functions*, classes. Atualmente, o ES6 encontra-se implementado, na maioria dos *browsers* atuais, entre 93% e 99%, exceto o Internet Explorer que se encontra a 11% [ecm], no entanto, o desenvolvimento posterior deste *browser* foi cancelado pela Microsoft.

Existe uma versão de EcmaScript 2016 (ES7), no entanto os *browsers* ainda estão a progredir para suportar a anterior.

Finalmente, o EcmaScript 2017 já começou a ser pensado, mas ainda se encontra numa fase muito inicial e a lista de melhorias ainda está a ser discutida.

2.4.6 Webpack

Webpack é um gestor de módulos. Um empacotador de módulos, ou *bundler*, pega em todos os módulos da aplicação e em todas as suas dependências e junta-os num ficheiro, ou vários, de forma a que a aplicação seja carregada mais rapidamente pelo *browser*, melhorando assim a *User Experience*. Para além disso, o Webpack, também dispõe de *loaders* que transformam as mais variadas linguagens para linguagens suportadas pelos *browsers* atuais, como por exemplo, de SCSS para CSS, de JSX para JS, e também no processo de converter JSX para JS, converte ao mesmo tempo, usando o *babel-loader*, ES6 para ES5. (Como descrito anteriormente, ES6 ainda não é suportado por todos os *browsers* mais antigos, pelo que é necessário utilizar uma versão de Javascript anterior, ES5).

Para além de empacotar, o Webpack também otimiza o código para o tornar mais leve e rápido. Para isto, porções inalcançáveis, dependências não utilizadas, comentários, são removidos, apenas restando o que é estritamente necessário, chamado *tree-shaking*. Por sua vez, o ficheiro resultante é *"minified"*, para melhores tempos de execução.

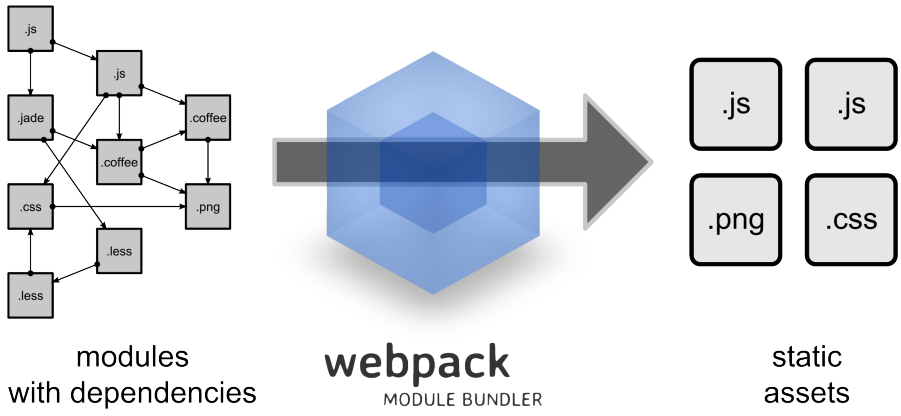

Figura 2.12: Webpack [wha]

Atualmente, enquanto esta dissertação está a ser escrita, está a ser desenvolvido o Webpack3, a nova versão de Webpack que promete ser mais rápido, mais estável, e com melhores tempos de execução.

2.4.7 Node e Express

NodeJs é uma plataforma do lado do servidor que foi construida sob o interpretador de JavaScript V8 da Google. O V8 é um interpretador de JS que compila, executa e gere código JavaScript

[goo]. Node foi criado em 2009 por Ryan Dahl. Esta plataforma é conhecida pelas suas *features* que a põem à frente como:

- É assíncrona e *event driven*, isto significa que nunca bloqueia quando é feito um pedido à API. Em termos práticos, significa que quando a API recebe um pedido, não tem que esperar pela resposta àquele pedido antes de receber outro pedido. Por exemplo, se a API recebesse dois pedidos, caso esta API fosse síncrona, tinha que receber o primeiro, responder e depois receber o segundo, e assim o segundo ia demorar mais tempo porque só podia ser processado após o primeiro. No caso de ser assíncrona, esta recebe o primeiro pedido e enquanto espera pelo processamento da responda, recebe o segundo. Pelo que, o segundo pedido pode ser processado mais rapidamente que o primeiro. Como não bloqueia é facilmente escalável.
- É muito rápida já que foi construida sob o motor V8.
- Não necessita de *buffers*. Isto porque envia os dados por partes.
- É multi plataforma, isto significa que pode ser executado tanto em máquinas Linux como Windows, como outras.

A definição oficial do NodeJs é:*"Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world."* [Fou]

No entanto, o Node, apesar de se desenvolver em JavaScript, é muito básico em termos de biblioteca, pelo que é necessário, para maior conformidade e rapidez de desenvolvimento, chamar bibliotecas ou *frameworks* externas. Isto é possível, já que foi desenvolvido um módulo chamado *npm* que torna a instalação de recursos externos muito fácil. Uma das mais conhecidas é a *framework* ExpressJS. Express é uma *framework*, que simplifica a criação de um serviço em Node [exp], já que contem módulos a que um *developer* pode recorrer para não ter que criar tudo de raiz. Alguns destes módulos são a configuração de *middleware* para respostas a pedidos *Hypertext Transfer Protocol* (*HTTP*) e definir roteamentos. No geral, ExpressJs é útil para reduzir o tempo da criação de um serviço *Representational State Transfer* (REST) em NodeJs.

2.5 Trabalhos relacionados

2.5.1 Dynamorse

Atualmente existe um trabalho similar (Dynamorse) ao abordado nesta dissertação. Este pode ser consultado no *GitHub* [SR16]. No entanto é demasiado complexo em que um utilizador sem experiência não o consegue compreender. Também não é visualmente apelativo. Daí a divergência entre esta solução proposta e o trabalho desenvolvido pela Streampunk.

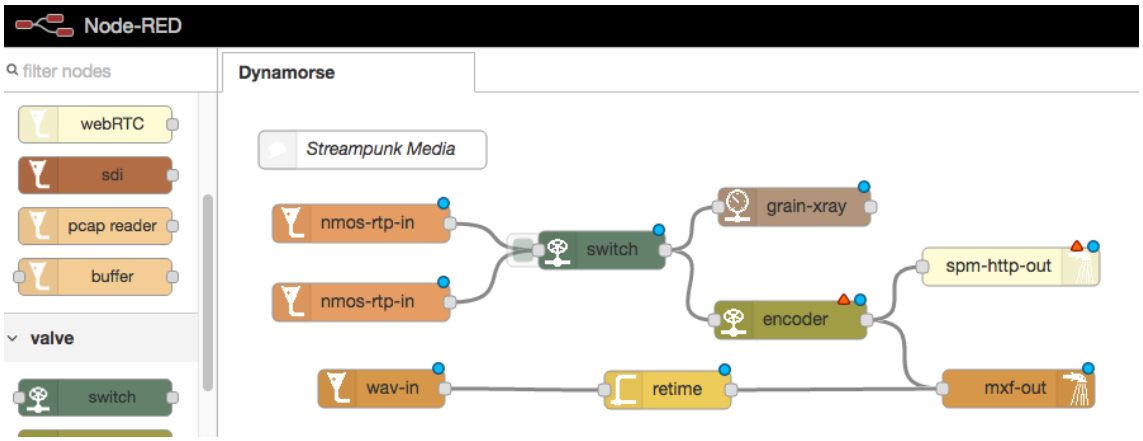

Figura 2.13: Exemplo da interface da Dynamorse [SR16]

Esta interface recorre ao *Node-RED* para a criação de diagramas [nod]. O *Node-RED* é uma ferramenta de programação visual, assim como esta dissertação. No entanto, o *Node-RED* não dispõem módulos de componentes com múltiplas portas, tanto de entrada como de saída e funciona a um nível de programação mais baixo, isto é, é possível criar várias abstrações que simplifiquem o trabalho do utilizador.

2.5.2 BBC

Atualmente, a BBC encontra-se a desenvolver um projeto que consiste no desenvolvimento de um estúdio sob IP. Foi testado um protótipo deste estúdio em R&D's trial at the 2014 Commonwealth Games [bbc15]. No entanto, a interface que eles criaram, apesar de funcional, é focada nas necessidades da BBC pelo que deixa de ser útil quanto aplicada a outras empresas de boradcast.

Pelo que foi necessário recorrer a uma alternativa para esta dissertação. Esta alternativa foi a biblioteca para React disponibilizada por Dylan Vorster e chamada "sorm-react-diagrams" [sto]. Esta biblioteca foi criada inicialmente pouco mais de um mês antes do inicio desta dissertação e foi desenvolvida aquando esta.

2.6 Resumo

Concluindo, podemos analisar que existem várias técnicas que a indústria televisiva recorre para fazer chegar o sinal às nossas casas. Também vimos que, para uma otimização de custos, os estúdios devem recorrer à atualização para tecnologias IP e *Cloud*. Algumas destas companhias já o estão a fazer (como é o caso da BBC [Raw15] e também a estudar qual a melhor maneira de a implementar, já que é uma alternativa nova ao modo como os estúdios funcionam.

Tudo indica que os estúdios televisivos num futuro próximo funcionem exclusivamente em *software* e quem desenvolver uma solução fiável e única estará à frente do mercado.

Para isto, esta dissertação foca-se numa parte do desenvolvimento desta solução.

O mundo web está em constante desenvolvimento e atualização, pelo que tecnologias atuais poderão encontrar-se obsoletas em pouco tempo caso não desenvolvam atualizações a acompanhar o mundo e as suas tendências. A maioria destas tecnologias são semelhantes, pelo que quando um *developer* escolhe a que tecnologias recorrer, esta decisão é, em parte, subjetiva, onde fatores como gostos e hábitos entram e influenciam a decisão. Por outro lado, é necessário ter em atenção os objetivos que existem para a aplicação a ser desenvolvida e verificar se as tecnologias vão de encontro a estes objetivos. Certas funcionalidades, que podem passar despercebidas podem tornar o trabalho do *developer* mais simples como muito mais complexo.

Capítulo 3

Solução proposta

Portanto, a solução proposta nesta dissertação é a criação de uma interface Web, na qual um técnico de um estúdio televisivo consiga criar o seu próprio estúdio televisivo personalizado à sua medida.

Esta interface terá que ser:

- Simples
- Atrativa
- Focada no utilizador e nas necessidades deste

A figura a seguir (figura 3.1) mostra uma possível representação de tal interface.

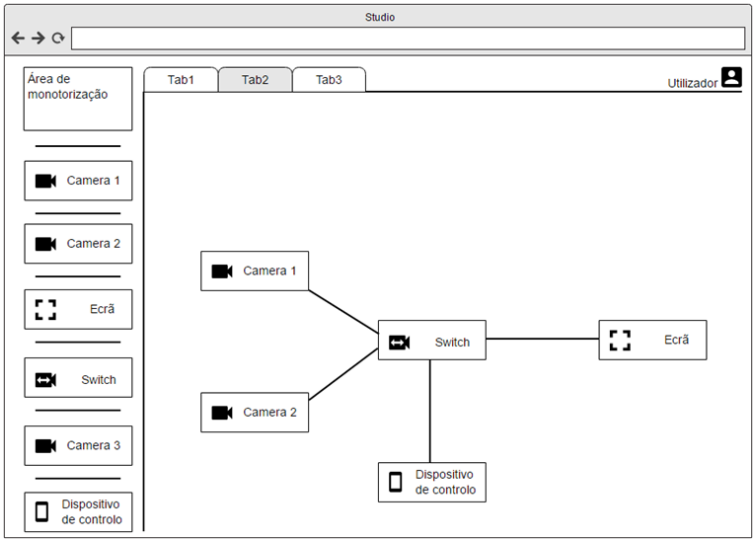

Figura 3.1: Mockup da interface onde existe um menu e um espaço de desenho

Neste *mockup* simplificado foi criado um estúdio constituído por duas câmaras que estão a transmitir sinal para um *switch* que por sua vez está a transmitir para um ecrã. Neste caso é possível controlar o *switch* através do dispositivo de controlo, que é uma aplicação num *smartphone*.

Do lado esquerdo encontram-se as várias componentes possíveis para criar o estúdio. Existem dois tipos de componentes: as estáticas e as dinâmicas. As estáticas são componentes que já existem independentemente dos dispositivos que se encontram na rede. Isto é, as componentes passíveis de virtualizar (passíveis de existirem apenas por *software* e deixarem de existir como equipamentos físicos). Estas podem variar de acordo com o sistema necessário. Por outro lado, as componentes dinâmicas são os dispositivos que se encontram na rede. Estes são encontrados através do serviço *multicast Domain Name System* (mDNS).

No centro do ecrã encontra-se um *canvas*, a parte principal desta dissertação. É aqui que um utilizador cria o seu estúdio. Este pode arrastar as componentes do lado esquerdo para este *canvas* e posteriormente fazer ligações ente elas. É de ter atenção que o número de entradas e saídas é alterado conforme a componente, por exemplo, uma câmara tem duas saídas, uma de áudio e uma de vídeo, enquanto o *switch* tem duas entradas, uma saída e um controlador. Um *switch* é a componente que de dois sinais de entrada permite-nos optar por um (que pode ser trocado em tempo-real) para transmitir como *output*. Neste *mockup* as entradas e saídas não estão devidamente especificados por uma razão de simplificação de leitura. Também é de ter atenção que existem restrições de ligação entre componentes, isto é, nem todas podem ser ligadas entre si diretamente, por exemplo, não podemos ligar uma câmara a um dispositivo de controlo.

Cada componente tem informação associada, como por exemplo, no caso de uma câmara, esta terá um IP, um operador, a qualidade em que transmite o vídeo, entre outras.

Também é importante referir que no canto superior esquerdo existe uma área de monitorização onde está a largura de banda a ser utilizada no momento. Cada componente consome largura de banda, no entanto o esquema criado pelo utilizador nunca poderá exceder a largura de banda existente na rede.

Um utilizador poderá criar vários diagramas devido à possível criação de separadores. Pode-se alternar entre separadores usando a barra superior do *mockup*. É útil para trabalhar com vários diagramas ao mesmo tempo ou então para trabalhar num diagrama muito complexo mais facilmente.

Por fim é criado um ficheiro com toda esta informação e enviado para um serviço que vai comunicar com cada uma destas componentes e ligá-las entre si virtualmente para que o estúdio seja criado. Este ficheiro é criado e enviado em tempo real sempre que haja uma alteração que resulte num esquema válido.

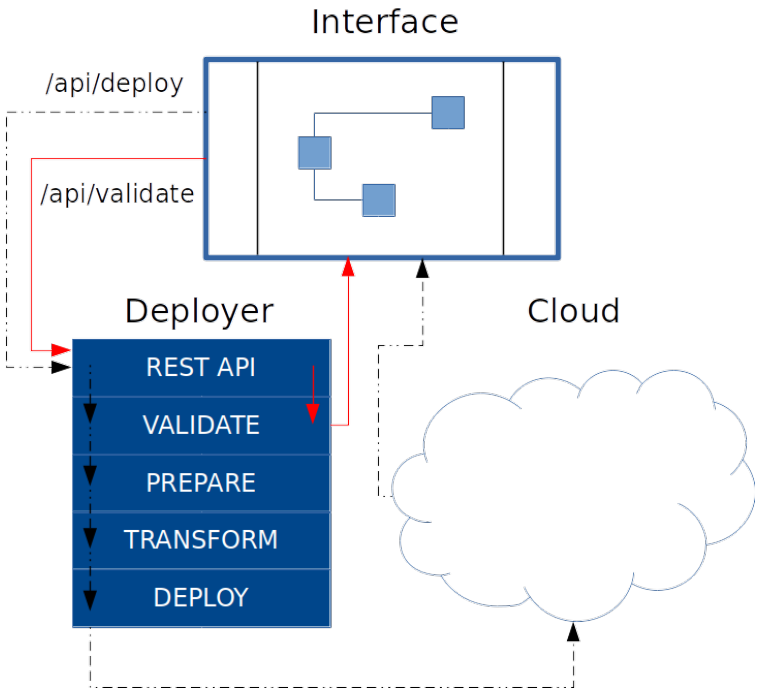

Figura 3.2: Arquitetura

Na figura 3.2 é apresentado um esquema da arquitetura. A Web App vai estar em constante comunicação com a API que por sua vez irá fazer pedidos à base de dados. Também a API irá comunicar com o orquestrador, enviando-lhe o ficheiro com toda a informação. Um objetivo secundário será que este orquestrador forneça um *feedback* que será útil à Web App. No entanto, como objetivo primário, esta comunicação apenas existe da API para o orquestrador.

3.1 Tecnologias

Para tal também temos que ter atenção às tecnologias a ser usadas. Estas serão, para o lado do cliente, ReactJS, HTML5, SCSS, ES6 e Webpack. Enquanto que para o lado do servidor serão usados NodeJS e ExpressJS.

Capítulo 4

A interface

Esta interface foi pensada de modo a ser o mais minimalista, útil e eficaz possível para os operadores do estúdio poderem virtualizar o seu estúdio televisivo. A figura a seguir exemplifica a criação de um simples estúdio.

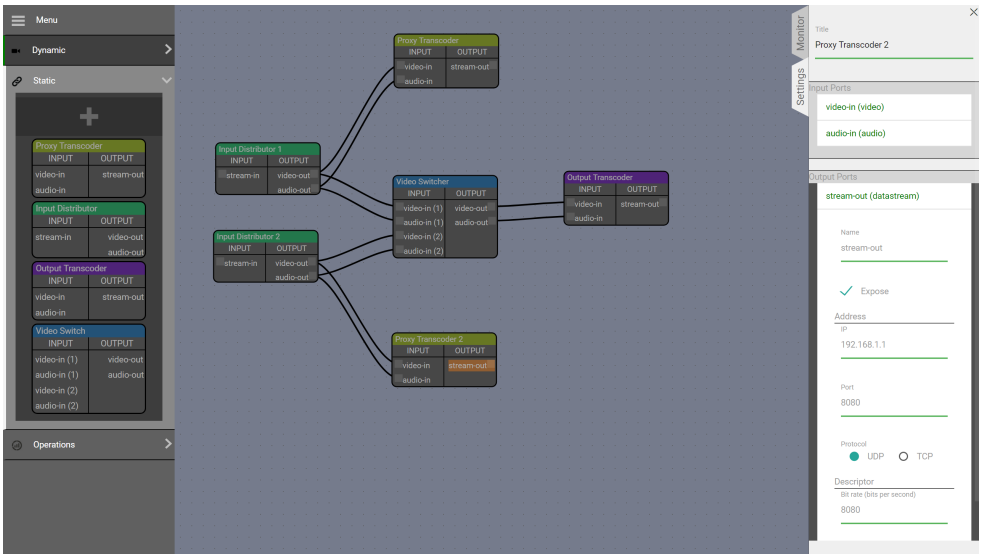

Figura 4.1: Exemplo de um diagrama válido construido através da interface.

Do lado esquerdo existe um menu onde existem componentes para escolher, no centro encontra-se o espaço de desenho e do lado direito vemos um menu onde é possível editar os parâmetros de cada componente.

4.1 Fluxo de informação

O resultado desta interface é a criação de um *JavaScript object notation* (*JSON*). *JSON* este que serve de meio de comunicação entre a interface e a dissertação do Vasco Gonçalves que, muito resumidamente, lança os componentes e as suas definições na [Gon17]. Este *JSON* foi desenvolvido em parceria com Vasco Gonçalves com atenção às necessidades de ambos. Também foi desenvolvido um *JSON Schema* para efeitos de validações. Tanto um exemplo de um *JSON* como o *JSON Schema* podem ser analisados nos anexos, respetivamente, A e B.

4.2 Um caso de estudo

Nesta secção irá ser demonstrado como funciona a interface sob um caso real. Vai ser exemplificado como criar e validar um simples diagrama. Existem dois utilizadores neste caso: o diretor técnico, que tem como responsabilidade criar e ligar as componentes estáticas; e o diretor televisivo, que é responsável por introduzir no diagrama as componentes dinâmicas (câmaras).

4.2.1 Diretor Técnico

Primeiro o diretor técnico abre a aplicação que é composta por uma tela vazia (ver figura 4.2).

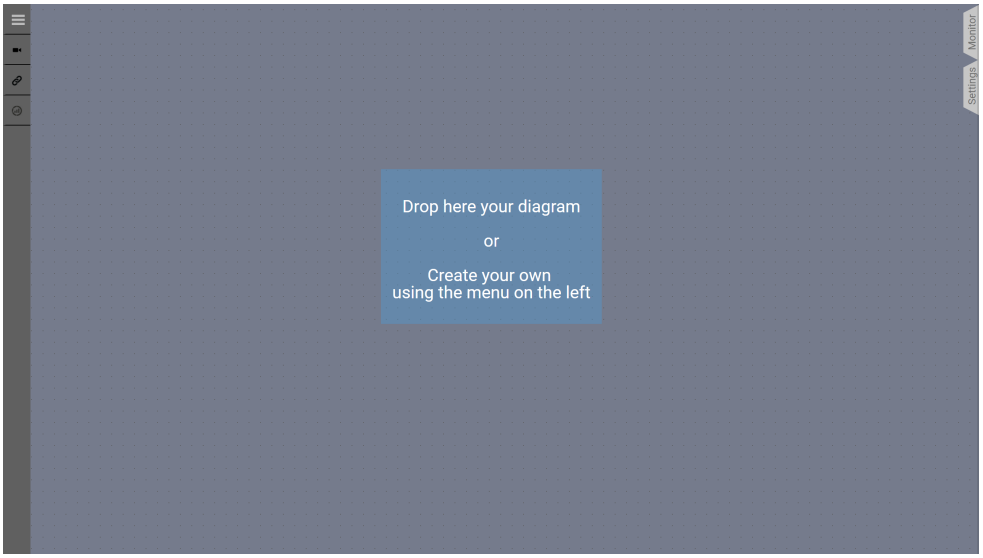

Figura 4.2: Vista inicial da interface

Aqui ele tanto pode criar o seu estúdio como carregá-lo ao arrastar um ficheiro para a tela. Caso este deseje criar um estúdio novo, pode fazê-lo recorrendo a ambos os menus (barra lateral esquerda e os dois botões à direita).

Apenas cabe ao diretor técnico criar o estúdio em si e ligar todos os dispositivos que necessita para o seu estúdio, exceto as câmaras, pelo que é da responsabilidade do diretor de televisão ou do produtor ligar estas últimas aos dispositivos do estúdio.

Portanto, tudo o que um diretor técnico necessita para criar o estúdio desejado é carregar no terceiro botão do menu do lado esquerdo e aí encontrará todos os dispositivos existentes para a criação do estúdio. (ver figura 4.3).

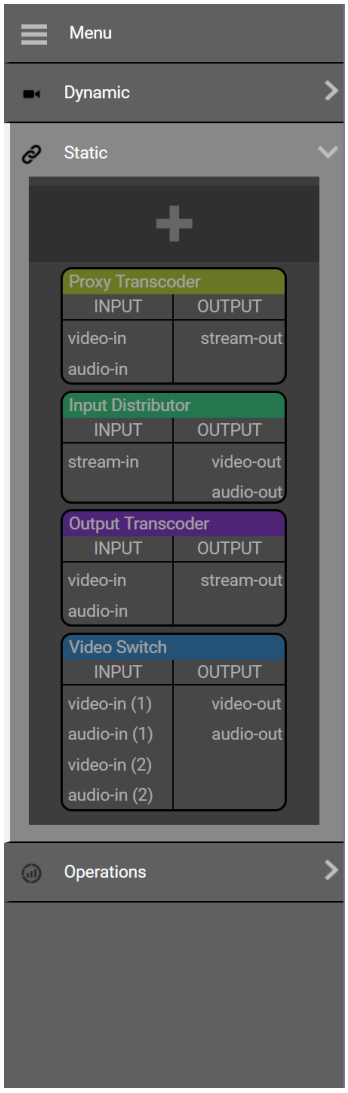

Figura 4.3: Menu lateral esquerdo. Vista sob a opção de componentes estáticos

Agora basta clicar nos componentes desejados e eles aparecerão na tela. Por sua vez, cada componente é composto por portas de entrada e portas de saída. É por estas portas que vai passar o vídeo e o áudio, logo é preciso definir alguns parâmetros como por exemplo: IP, número da porta, protocolo, dimensões do vídeo, entre outros. Para isto, o diretor técnico recorre ao botão *Settings* ou apenas clica sobre a porta que deseja editar e um menu do lado direito abre-se com toda a informação sobre cada componente. (ver figura 4.1). Para facilitar o trabalho do diretor técnico e para minimizar o tempo da criação do estúdio, este pode ligar portas de saída a portas de entrada e assim os valores dessa ligação serão sempre os mesmos.

Caso o utilizador cometa algum erro, por exemplo algum valor de uma porta inválido, ou uma ligação inválida, a interface notifica-o de modo a que possa ser corrigido o mais rapidamente possível (ver figuras 4.4 e 4.5).

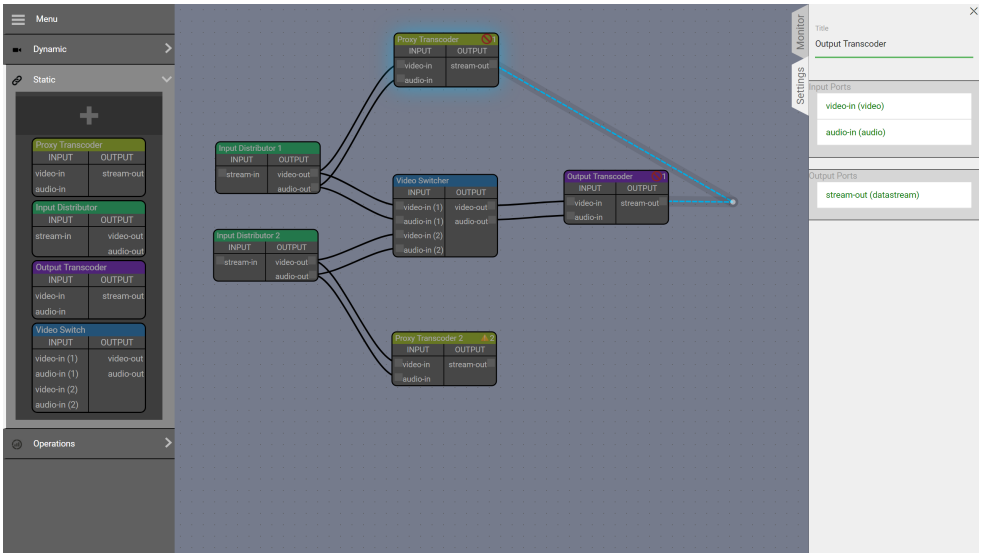

Figura 4.4: Exemplo de uma ligação inválida.

Na figura acima verificamos uma ligação inválida porque duas portas de saída nunca podem estar ligadas. Ligação entre *Proxy Transcoder* e *Output Transcoder*. Erro representado por um sinal de proibido na componente em causa e o número adjacente do lado direito representa o número de erros de ligações que o componente tem.

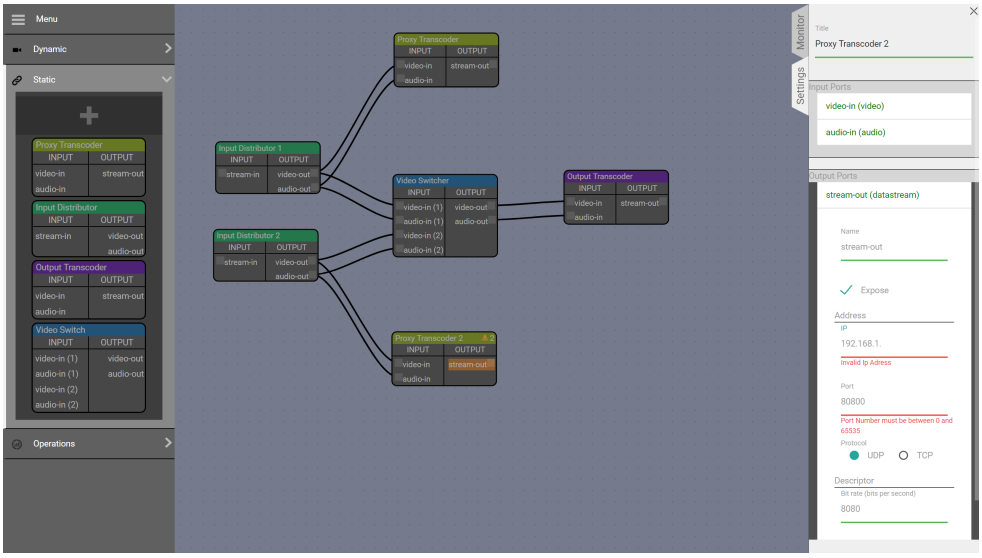

Figura 4.5: Exemplo de parâmetros de uma porta inválidos.

Na figura acima verificamos um erro de parâmetros inválidos. A porta em causa está evidenciada a laranja. Para notificar o utilizador é apresentada uma mensagem a explicar como resolver o erro por baixo de cada campo. Também é apresentado na componente um sinal de perigo a amarelo e um número a identificar o número de erros existentes.

Após todos os erros estarem resolvidos, o diretor técnico confirma que o estúdio que acabou de desenhar é válido utilizando a opção *Validate* do menu *Operations* e caso seja, guarda-o no seu computador, recorrendo à opção *Download* (ver figura 4.6).

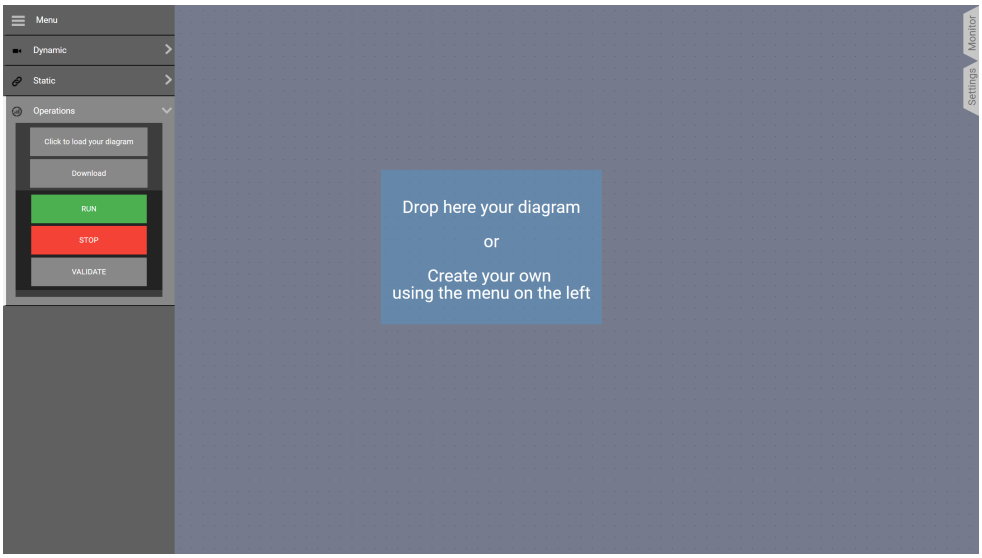

Figura 4.6: Vista sob a opção de operações do menu lateral do lado esquerdo

O trabalho do diretor técnico fica assim concluído. Este tem como objetivo criar uma biblioteca de estúdios para que serão utilizados posteriormente pelo diretor televisivo e o produtor.

4.2.2 Diretor Televisivo

Agora entra o diretor televisivo. Este apenas tem que fazer *Upload* do estúdio criado pelo diretor técnico, ligar as câmaras ao sistema e correr ou parar o sistema. Para ligar as câmaras existem duas alternativas.

A primeira é utilizando o menu *Dynamic* que procura na rede todas as câmaras registadas por mDNS. Nesta alternativa, muito mais intuitiva para o utilizador, este apenas tem que criar uma ligação entre a câmara e o componente desejado.

A outra alternativa, caso a câmara não se encontre registada por mDNS ou o sistema não a encontre, o diretor televisivo pode manualmente introduzir, nas entradas válidas dos componentes desejados , os endereços IP e as portas das câmaras.

Em ambos os casos, o diretor televisivo deve sempre garantir que o estúdio criado é válido utilizando a opção *Validate*. Caso seja válido pode então virtualizar o estúdio ou pará-lo utilizando as opções a verde e a vermelho da figura 4.6.

Finalmente pode monitorizar o sistema como um todo (ver figura 4.7) ou cada componente individualmente (ver figuras 4.8 e 4.9). Esta monitorização é feita a nível de consumos de CPU, memória, largura de banda, ...

Figura 4.7: Exemplo de monitorização, em tempo real, do sistema como um todo, recorrendo ao Grafana

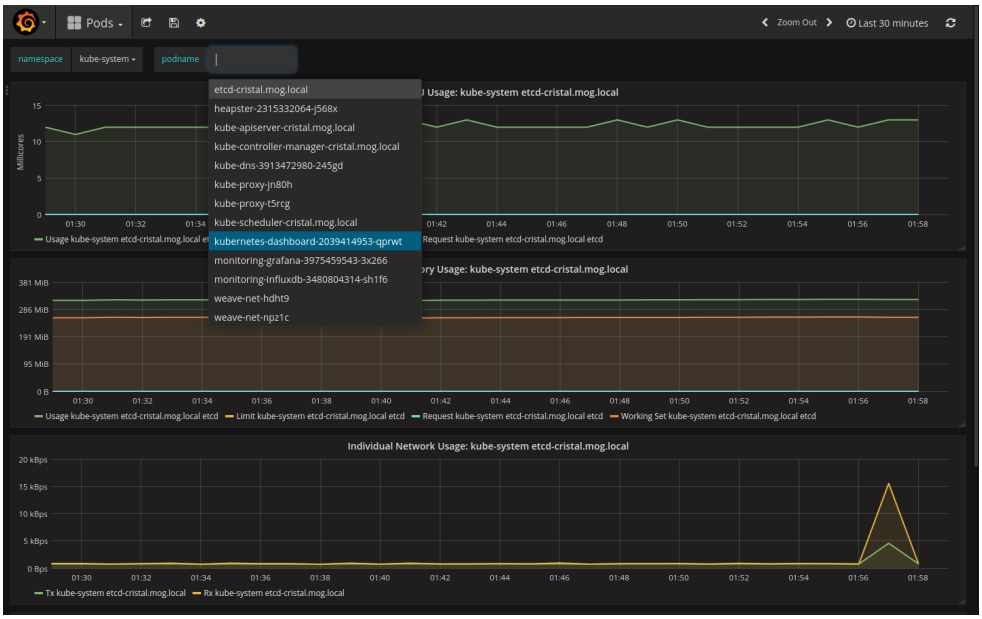

Figura 4.8: Exemplo de monitorização individual - escolha da componente, recorrendo ao Grafana

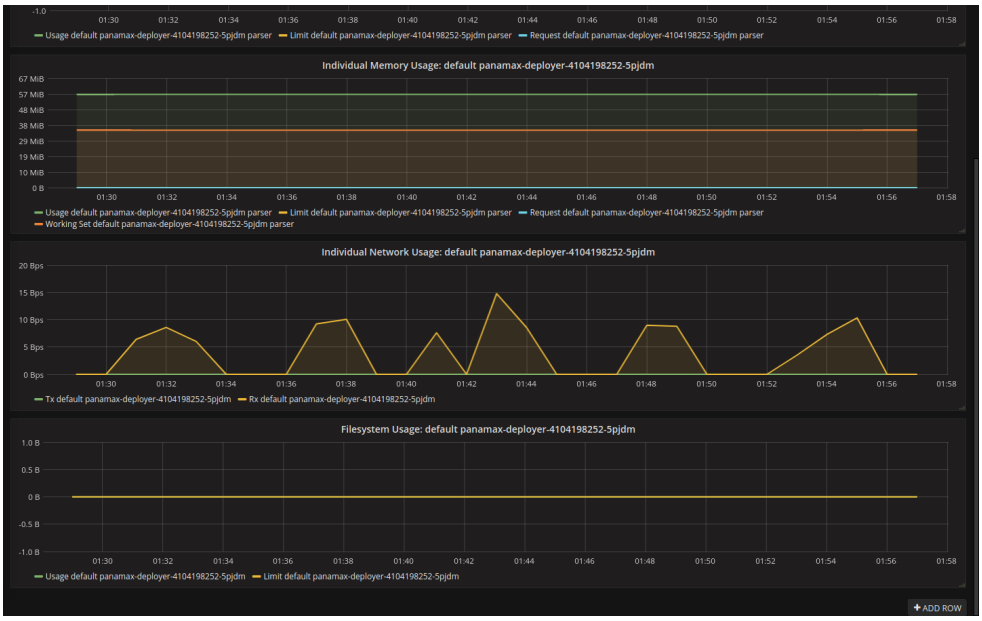

Figura 4.9: Exemplo de monitorização, em tempo real, de um componente específico do sistema, recorrendo ao Grafana

4.3 Avaliação e Testes

Uma parte importante para o desenvolvimento de uma interface é a sua avaliação e testes. Foram realizados testes de usabilidade e testes A/B. Estes testes foram realizados apenas por trabalhadores da MOG. Utilizadores habituados às necessidades de um estúdio televisivo. Estes utilizadores tiveram que criar um estúdio válido e também foram mostradas interfaces diferentes para avaliar em qual seria mais intuitivo criar o estúdio.

Testes de usabilidade são úteis para avaliar se um utilizador trabalhar com a interface de uma forma intuitiva, e rápida. Também são úteis para analisar se o utilizador necessita de mais algum menu ou opção. O resultado obtido neste teste foi que deveria ser realizada uma alteração do menu de alteração de propriedades das componentes, evidenciado pela figura 4.10) . Que dantes era um modal. Esta alteração foi necessária porque os utilizadores tinham a necessidade de ver as componentes e as suas ligações enquanto editavam as propriedades de uma certa componente. Para isso, o diagrama tinha que estar visível, o que não acontecia quando existia um modal à frente.

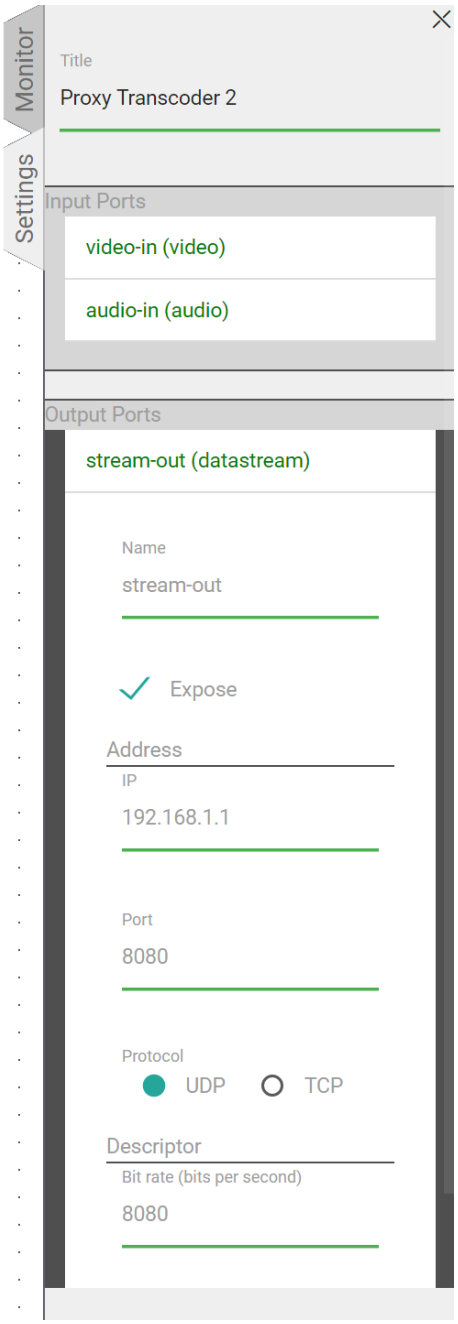

Figura 4.10: Menu de edição das propriedades das componentes

Por outro lado, os testes A/B são úteis para avaliar se a interface está apelativa. Com recurso a este tipo de testes, o item mais sujeito a alterações foi o que representa as componentes. Pode ser visualizado um exemplo final destas componentes na figura 4.11). Esta evoluiu para algo mais fácil de perceber e mais funcional.

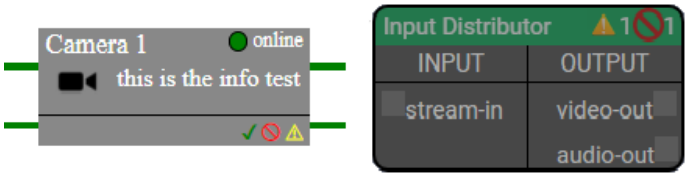

Figura 4.11: Do lado esquerdo verificamos uma componente antes de testes A/B serem realizados e do lado direito após

Com o recurso a estes dois testes, especialmente o teste A/B durante o desenvolvimento e o teste de usabilidade na parte final do projeto, a interface sofreu várias alterações. No entanto estas alterações não estão totalmente completas. Um outro resultado que os testes forneceram é que os utilizadores, intuitivamente, ao passar uma componente do menu para o *canvas*, onde é criado o diagrama, os utilizadores tendem a carregar na componente que desejam e arrastam, no entanto a interface não permite arrastar a partir do menu, apenas clicar e o componente aparece no *canvas*. Este é um aspeto que terá que ficar corrigido no futuro para uma melhor usabilidade.

Capítulo 5

Trabalho futuro e conclusão

Este é um tipo de trabalho onde as melhorias podem, muitas vezes, parecer incalculáveis tanto a nível de funcionalidades como de estilo e *user experience*. O projeto está funcional para ir para testes pelo mercado, isto é testes por empresas de broadcast para avaliar se é necessário alguma diferente funcionalidade, no entanto, alguns aspetos como o *feedback* dado pelo sistema e apresentado ao utilizador está passível de ser melhorado com: mensagens mais específicas e em diferentes linguagens; quando uma componente de um diagrama que está a correr falhe por qualquer razão, seja apresentado um *warning* ao utilizador que explique o erro e como o corrigir. Também a possibilidade de criar uma componente customizada não ficou completa na sua totalidade. Uma componente customizada significa que o utilizador pode escolher tanto os campos desta como o número de portas de saída e entrada e seus tipos.

Seria útil acrescentar etiquetas e setas de direção às ligações das componentes. Isto para ser mais fácil ao utilizador entender o sentido do fluxo e identificar ligações.

Uma outra melhoria seria permitir, ao utilizador, escolher a posição dos menus (topo, direita, baixo, esquerda). Isto para que a ferramenta esteja personalizada aos hábitos do utilizador, sendo assim mais fácil de utilizar.

Por fim, mais uma possível melhoria seria acrescentar separadores (como apresentados nos *mockups* do capítulo 3) para que o utilizador possa trabalhar em dois ou mais diagramas ao mesmo tempo e ligá-los entre si tanto para conveniência como para que possa estar a criar um diagrama complexo muito mais facilmente. No entanto, a interface permite tanto *zoom-in* e *out* como arrastamento, pelo que é possível criar esquemas complexos desta maneira. Neste caso os separadores serviam para o utilizador ter mais uma escolha de como organizar o esquema.

A aplicação de tecnologias *Cloud* e IP para estúdios televisivos traz consigo muitas vantagens como, por exemplo, a grande diminuição no número de carrinhas estúdio e a obsolescência dos

componentes específicos num estúdio. Algumas das componentes específicas de um estúdio são observadas no capítulo 4. Atualmente, estas componentes estão fisicamente presentes num estúdio. Ao virtualizar estas componentes existem duas alternativas para um estúdio televisivo: ou aluga, ou compra um serviço na *Cloud* onde as componentes, previamente físicas, estão a ser virtualizadas por *software*, ou então compra as máquinas físicas que virtualizem as componentes. A vantagem de ter as componentes virtualizadas é que quando uma determinada componente não é necessária no momento, a máquina, onde esta está a correr, pode eliminar a componente e lançar outra qualquer. Enquanto que se as componentes fossem físicas, e não fossem necessárias, iam estar paradas num estúdio, a ocupar espaço valioso, pelo que o estúdio iria perder dinheiro com estas. Tudo isto é extremamente útil para diminuir as despesas de um estúdio.

Esta dissertação abordou uma ponte entre os mundos de *Broadcast* e IP/*Cloud*. Esta ponte é uma interface Web onde é possível criar e virtualizar um estúdio televisivo. É uma interface simples e fácil de utilizar por um técnico de estúdio sem mais qualificações do que as necessárias para trabalhar num estúdio televisivo.

Foram atingidos todos os objetivos primários e alguns dos secundários previstos.

Com esta dissertação aprendi em profundidade como funciona um estúdio televisivo e o mundo de *Broadcast*. Expandi, também, os meus conhecimentos sobre tecnologias Web e o desenvolvimento de interfaces.

Atualmente, para um técnico televisivo criar ou editar o seu estúdio ia ter que fisicamente trocar os dispositivos e os cabos ligados entre eles. Com esta interface, o mesmo técnico pode fazer tudo isto sem ter que se deslocar aos dispositivos, sendo que não necessita de sair do seu computador, ou até da sua casa.

Anexo A

JSON Exemplo

```
{
  "containers": [
    {
      "name": "input1",
      "image": "docker.mog.local:5000/inputdistributor:0.3-vgoncalves",
      "pins": {
        "in": [
          {
            "name": "stream-in",
            "expose": true,
            "address": {
              "ip": "127.0.0.1",
              "port": {
                "protocol": "UDP",
                "number": 2000
              }
            },
            "descriptor": {
              "type": "datastream",
              "data": {
                "bitrate": 2402000
              }
            }
          }
        ],
        "out": [
          {
            "name": "video-out",
            "expose": false,
            "address": {
              "ip": "225.2.2.0",
              "port": {
                "protocol": "UDP",
```

```
                "number": 3000
              }
            },
            "descriptor": {
              "type": "video",
              "data": {
                "width": 176,
                "height": 100,
                "framerate": 25.0,
                "colordepth": 8,
                "colorspace": "4:2:0",
                "interlaced": false
              }
            }
          }
        ]
      },
      "envs": {
        "inPort": "2000",
        "multicastAddrIP": "225.2.2.0",
        "multicastAddrPort": 3000,
        "videoHeight": "100",
        "videoWidth": "176",
        "waitingTime": "30"
      }
    },
    {
      "name": "input2",
      "image": "docker.mog.local:5000/inputdistributor:0.3-vgoncalves",
      "pins": {
        "in": [
          {
            "name": "stream-in",
            "expose": true,
            "address": {
              "ip": "127.0.0.1",
              "port": {
                "protocol": "UDP",
                "number": 2000
              }
            },
            "descriptor": {
              "type": "datastream",
              "data": {
                "bitrate": 2402000
              }
            }
          }
        ],
```

```
        "out": [
          {
            "name": "video-out",
            "expose": false,
            "address": {
              "ip": "225.2.2.2",
              "port": {
                "protocol": "UDP",
                "number": 3000
              }
            },
            "descriptor": {
              "type": "video",
              "data": {
                "width": 176,
                "height": 100,
                "framerate": 25.0,
                "colordepth": 8,
                "colorspace": "4:2:0",
                "interlaced": false
              }
            }
          }
        ]
      },
      "envs": {
        "inPort": "2000",
        "multicastAddrIP": "225.2.2.2",
        "multicastAddrPort": 3000,
        "videoHeight": "100",
        "videoWidth": "176",
        "waitingTime": "30"
      }
    },
    {
      "name": "proxy1",
      "image": "docker.mog.local:5000/proxytranscoder:0.3-vgoncalves",
      "pins": {
        "in": [
          {
            "name": "video-in",
            "expose": false,
            "address": {
              "ip": "225.2.2.0",
              "port": {
                "protocol": "UDP",
                "number": 3000
              }
            },
```

```
            "descriptor": {
              "type": "video",
              "data": {
                "width": 176,
                "height": 100,
                "framerate": 25.0,
                "colordepth": 8,
                "colorspace": "4:2:0",
                "interlaced": false
              }
            }
          }
        ],
        "out": [
          {
            "name": "dash-out",
            "expose": true,
            "address": {
              "ip": "127.0.0.1",
              "port": {
                "protocol": "TCP",
                "number": 80
              }
            },
            "descriptor": {
              "type": "other",
              "data": {}
            }
          }
        ]
      },
      "envs": {
        "multicastAddrIP": "225.2.2.0",
        "multicastAddrPort": 3000,
        "videoHeight": "100",
        "videoWidth": "176"
      }
    },
    {
      "name": "proxy2",
      "image": "docker.mog.local:5000/proxytranscoder:0.3-vgoncalves",
      "pins": {
        "in": [
          {
            "name": "video-in",
            "expose": false,
            "address": {
              "ip": "225.2.2.2",
              "port": {
```

```
                "protocol": "UDP",
                "number": 3000
              }
            },
            "descriptor": {
              "type": "video",
              "data": {
                "width": 176,
                "height": 100,
                "framerate": 25.0,
                "colordepth": 8,
                "colorspace": "4:2:0",
                "interlaced": false
              }
            }
          }
        ],
        "out": [
          {
            "name": "dash-out",
            "expose": true,
            "address": {
              "ip": "127.0.0.1",
              "port": {
                "protocol": "TCP",
                "number": 80
              }
            },
            "descriptor": {
              "type": "other",
              "data": {}
            }
          }
        ]
      },
      "envs": {
        "multicastAddrIP": "225.2.2.2",
        "multicastAddrPort": 3000,
        "videoHeight": "100",
        "videoWidth": "176"
      }
    },
    {
      "name": "output",
      "image": "docker.mog.local:5000/outputtranscoder:0.3-vgoncalves",
      "pins": {
        "in": [
          {
            "name": "video-in",
```

```
            "expose": false,
            "address": {
              "ip": "225.2.2.2",
              "port": {
                "protocol": "UDP",
                "number": 3000
              }
            },
            "descriptor": {
              "type": "video",
              "data": {
                "width": 176,
                "height": 100,
                "framerate": 25.0,
                "colordepth": 8,
                "colorspace": "4:2:0",
                "interlaced": false
              }
            }
          }
        ],
        "out": [
          {
            "name": "video-out",
            "expose": false,
            "address": {
              "ip": "10.0.0.254",
              "port": {
                "protocol": "UDP",
                "number": 4000
              }
            },
            "descriptor": {
              "type": "video",
              "data": {
                "width": 176,
                "height": 100,
                "framerate": 25.0,
                "colordepth": 8,
                "colorspace": "4:2:0",
                "interlaced": false
              }
            }
          }
        ]
      },
      "envs": {
        "multicastAddrIP": "225.2.2.0",
        "multicastAddrPort": 3000,
```

JSON Exemplo

```
        "videoHeight": "100",
        "videoWidth": "176",
        "outIP": "pc-vgoncalves.mog.local",
        "outPort": "4000"
      }
    }
  ],
  "cameras": [],
  "connections": []
}
```

Anexo B

JSON Schema

```
{
  "definitions": {
    "pin": {
      "title": "Pin",
      "type": "object",
      "properties": {
        "name": {
          "type": "string",
          "pattern": "^[a-z0-9]([-a-z0-9]*[a-z0-9])?$"
        },
        "expose": {
          "type": "boolean"
        },
        "address": {
          "type": "object",
          "properties": {
            "ip": {
              "type": "string",
              "pattern": "^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.)
                  {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$"
            },
            "port": {
              "type": "object",
              "properties": {
                "protocol": {
                  "enum": [
                    "TCP",
                    "UDP"
                  ]
                },
                "number": {
                  "type": "integer",
                  "minimum": 0,
```

```
                "maximum": 65535
              }
            },
            "required": [
              "protocol",
              "number"
            ]
          }
        },
        "required": [
          "ip",
          "port"
        ]
      },
      "descriptor": {
        "$ref": "#/definitions/descriptor"
      }
    },
    "required": [
      "name",
      "expose",
      "address",
      "descriptor"
    ]
  },
  "descriptor": {
    "title": "Descriptor",
    "type": "object",
    "oneOf": [
      {
        "properties": {
          "type": {
            "enum": ["video"]
          },
          "data": {
            "type": "object",
            "properties": {
              "width": {"type": "integer", "minimum":0},
              "height": {"type": "integer", "minimum":0},
              "framerate": {"type": "number", "minimum":0},
              "colordepth": {"type": "integer", "minimum":0},
              "colorspace": {"enum": ["4:4:4","4:2:2","4:2:1","4:1:1","4:2:0","
                  4:1:0","3:1:1"]},
              "interlaced": {"type": "boolean"}
            },
            "required": ["width","height","framerate","colordepth","colorspace","
                interlaced"]
          }
        }
```

```
        },
        {
          "properties": {
            "type": {
              "enum": ["audio"]
            },
            "data": {
              "type": "object",
              "properties": {
                "samplerate": {"type": "integer", "minimum":0},
                "bitdepth": {"type": "integer", "minimum":0},
                "channels": {"type": "integer", "minimum":0}
              },
              "required": ["samplerate","bitdepth","channels"]
            }
          }
        },
        {
          "properties": {
            "type": {
              "enum": ["datastream"]
            },
            "data": {
              "type": "object",
              "properties": {
                "bitrate": {"type": "integer", "minimum":0}
              },
              "required": ["bitrate"]
            }
          }
        },
        {
          "properties": {
            "type": {
              "enum": ["other"]
            },
            "data": {
              "type": "object"
            }
          }
        }
      ],
      "required": ["type","data"]
    },
    "pins": {
      "title": "Pins",
      "type": "object",
      "properties": {
        "in": {
```

```
          "type": "array",
          "items": {
            "$ref": "#/definitions/pin"
          }
        },
        "out": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/pin"
          }
        }
      },
      "required": [
        "in",
        "out"
      ]
    },
    "container": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        },
        "image": {
          "type": "string"
        },
        "pins": {
          "$ref": "#/definitions/pins"
        },
        "envs": {
          "type": "object"
        },
        "iface": {
          "$ref": "#/definitions/iface/modules"
        }
      },
      "required": [
        "name",
        "image",
        "pins",
        "envs"
      ]
    },
    "containers": {
      "title": "Containers",
      "type": "array",
      "items": {
        "$ref": "#/definitions/container"
      }
```

```
    },
    "camera": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        },
        "type": {
          "type": "string"
        },
        "pins": {
          "$ref": "#/definitions/pins"
        },
        "iface": {
          "$ref": "#/definitions/iface/modules"
        }
      },
      "required": [
        "name",
        "type",
        "pins"
      ]
    },
    "cameras": {
      "title": "Cameras",
      "type": "array",
      "items": {
        "$ref": "#/definitions/camera"
      }
    },
    "connection": {
      "title": "Connection",
      "type": "object",
      "properties": {
        "module": {
          "type": "string"
        },
        "pin": {
          "type": "string"
        }
      },
      "required": [
        "module",
        "pin"
      ]
    },
    "connections": {
      "title": "Connections",
      "type": "array",
```

```
      "items": {
        "type": "object",
        "properties": {
          "source": {
            "$ref": "#/definitions/connection"
          },
          "destination": {
            "$ref": "#/definitions/connection"
          },
          "iface": {
            "$ref": "#/definitions/iface/connections"
          }
        },
        "required": [
          "source",
          "destination"
        ]
      }
    },
    "iface" : {
      "position": {
        "title": "Position",
        "description": "Position on interface",
        "type": "object",
        "properties": {
          "x": {
            "type": "number"
          },
          "y": {
            "type": "number"
          }
        },
        "required": [
          "x",
          "y"
        ]
      },
      "modules": {
        "title": "Modules Interface",
        "description": "Interface data for modules",
        "type": "object",
        "properties": {
          "position": {
            "$ref": "#/definitions/iface/position"
          }
        },
        "required": [
          "position"
        ]
```

```
      },
      "connections": {
        "title": "Connections Interface",
        "description": "Interface data for connections",
        "type": "object",
        "properties": {
          "points": {
            "type": "array",
            "items": {
              "$ref": "#/definitions/iface/position"
            }
          }
        },
        "required": [
          "points"
        ]
      }
    }
  },
  "type": "object",
  "properties": {
    "containers": {
      "$ref": "#/definitions/containers"
    },
    "cameras": {
      "$ref": "#/definitions/cameras"
    },
    "connections": {
      "$ref": "#/definitions/connections"
    }
  },
  "required": [
    "containers",
    "cameras",
    "connections"
  ]
}
```

Referências

[ADJ+11] Shay Artzi, Julian Dolby, Simon Holm Jensen, Anders Moller e Frank Tip. A framework for automated testing of javascript web applications. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 571–580. IEEE, 2011.

[bbc15] The network behind the r&d commonwealth games 2014 showcase - bbc r&d, Nov 2015. URL: http://www.bbc.co.uk/rd/blog/2014-07-commonwealth-games-showcase-network.

[BBD+04] K Bacic, M Boban, D Defar, V Klepac, M Attree e H Balasko. Digital TV outside broadcast vehicle. In *Electronics in Marine, 2004. Proceedings Elmar 2004. 46th International Symposium*, pages 449–454. IEEE, 2004.

[BRWT13] P J Brightwell, J D Rosser, R N J Wadge e P N Tudor. The IP Studio. 2013. URL: http://digital-library.theiet.org/docserver/fulltext/conference/ibc2013/20130016.pdf?expires=1486570888{&}id=id{&}accname=guest{&}checksum=C51AF8B9044B902098300B18A65B24E2.

[Bub16] Nataliia Bubniuk. Comparison of js frameworks: Angularjs vs. reactjs vs. ember.js - dzone web dev, Nov 2016. URL: https://dzone.com/articles/comparison-of-js-frameworks-angularjs-vs-reactjs-v.

[css17] Leveling up with react: Redux, Apr 2017. URL: https://css-tricks.com/learning-react-redux/.

[dCP16] Miguel Ferreira da Cunha Poeira. Virtualização de estúdios móveis na produção de conteúdos audiovisuais em direto. 2016.

[ecm] Ecmascript 6 compatibility table. URL: http://kangax.github.io/compat-table/es6/.

[exp] Express - node.js web application framework. URL: http://expressjs.com/.

[Fou] Node.js Foundation. Node.js. URL: https://nodejs.org/en/.

[fut14] May i introduce you to "sassy css"?, Jun 2014. URL: https://futurestud.io/tutorials/may-i-introduce-you-to-sassy-css.

[Gal07] W.O. Galitz. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. Wiley Desktop Editions. Wiley, 2007. URL: https://books.google.pt/books?id=Q3Xp_Awu49sC.

REFERÊNCIAS

[Gar10] J.J. Garrett. *Elements of User Experience,The: User-Centered Design for the Web and Beyond*. Voices That Matter. Pearson Education, 2010. URL: https://books.google.pt/books?id=9QC6r5OzCpUC.

[Gon17] Vasco Gonçalves. *Orquestração e aprovisionamento de um estúdio televisivo baseado na tecnologia IP na Cloud*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, 2017.

[goo] Chrome v8 | google developers. URL: https://developers.google.com/v8/.

[Jac14] Jim Jachetta. Ip to the camera-completing the broadcast chain. In *Annual Technical Conference & Exhibition, SMPTE 2014*, pages 1–29. SMPTE, 2014.

[Jor05] Jacob W Jorgensen. Transmission control protocol/internet protocol (tcp/ip) packet-centric wireless point to multi-point (ptmp) transmission system architecture, March 1 2005. US Patent 6,862,622.

[lib11] Library vs. framework?, Sep 2011. URL: http://www.programcreek.com/2011/09/what-is-the-difference-between-a-java-library-and-a-framework/.

[Net16] Cisco Visual Networking. Cisco Global Cloud Index: Forecast and Methodology 2015-2020 (White Paper), 2016. URL: http://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf.

[nod] Node-red. URL: https://nodered.org/.

[OMO11] Hyoung-Yong Oh, Byoung-Won Min e Yong-Sun Oh. User Customized Web Interface Design Optimized for SaaS-based Digital Library System-focusing on the LinkSaaS Website. *The Journal of the Korea Contents Association*, 11(5):148–156, 2011. URL: http://ocean.kisti.re.kr/downfile/volume/kocon/CCTHCV/2011/v11n5/CCTHCV{_}2011{_}v11n5{_}148.pdf.

[Opt17] Optimizely. Optimization glossary: A/b testing, 2017. URL: https://www.optimizely.com/ab-testing/.

[PS09] Vishnu S Pendyala e Simon S Y Shim. The Web as the ubiquitous computer. *IEEE Computer*, 42(9):90–92, 2009. URL: https://72198858-a-62cb3a1a-s-sites.googlegroups.com/site/ssdd5068/web{_}ubiquos.pdf?attachauth=ANoY7crw5d-ksfIfA88oJT0-140I3u4evno1ZugDEuf2-RlAkUgfB{_}MVKHgHfVjA8I-lGKONh0.

[Raw15] Alexander Rawcliffe. Covering the Glasgow 2014 Commonwealth Games using IP Studio. (March), 2015.

[RUS15] KATHERINE RUSHTON. How we spend a decade of our adult life watching tv: Average briton now spends 24 hours a week in front of the box, jul 2015. URL: http://www.dailymail.co.uk/news/article-3179425/How-spend-decade-adult-life-watching-TV-Average-Briton-spends-24-hours-week.html.

[Ser16] Olyimpic Broadcasting Services. OBS Media Fact File. 2016. URL: https://www.obs.tv/private/pbdocs/Rio2016-OBSMediaFactFile-16112016.pdf.

[SR16] Rogers Simon e Cartwright Richard. Dynamorse. https://github.com/Streampunk/dynamorse, 2016.

[sto]

[Tec16] TechFAQ. How television broadcasting works, mar 2016. URL: http://www.tech-faq.com/how-television-broadcasting-works.html.

[VWO17] VWO. The complete guide to a/b testing, 2017. URL: https://vwo.com/ab-testing/.

[wha] Webpack. URL: https://webpack.github.io/docs/what-is-webpack.html.